



# PROGRAMACIÓN II – UNIDAD 2

## PYTHON 2

# Lenguaje python



## Tupla

Una tupla es una colección ordenada e **inmutable** . En Python, las tuplas se escriben entre paréntesis.

# Lenguaje python

## Tuplas

Todo lo que hemos explicado sobre las listas se aplica también a las tuplas, a excepción de la forma de definirla, para lo que se utilizan paréntesis en lugar de corchetes.

```
t = (1, 2, True, "python")
```

En realidad el constructor de la tupla es la coma, no el paréntesis, pero el intérprete muestra los paréntesis, y nosotros deberíamos utilizarlos, por claridad.

```
>>> t = 1, 2, 3
>>> type(t)
type "tuple"
```

# Lenguaje python - Tuplas

Además hay que tener en cuenta que es necesario añadir una coma para tuplas de un solo elemento, para diferenciarlo de un elemento entre paréntesis.

```
>>> t = (1)
>>> type(t)
type "int"
>>> t = (1,)
>>> type(t)
type "tuple"
```

# Lenguaje python - Tuplas

Para referirnos a elementos de una tupla, como en una lista, se usa el operador []:

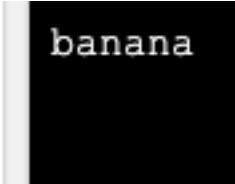
```
mi_var = t[0] # mi_var es 1  
mi_var = t[0:2] # mi_var es (1, 2)
```

# Lenguaje python - Tuplas

## Acceder a elementos de tupla

Puede acceder a elementos de tupla consultando el número de índice, entre corchetes:

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple[1])
```

A black rectangular box with a thin white border on the left side, containing the word "banana" in white text, representing the output of the Python code.

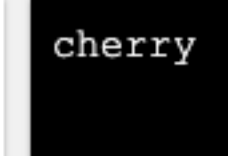
banana

# Lenguaje python - Tuplas

## Indexación negativa

La indexación negativa significa comenzar desde el final, se -1 refiere al último elemento, se -2 refiere al penúltimo elemento, etc.

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple[-1])
```



cherry

# Lenguaje python - Tuplas

## Rango de índices

Puede especificar un rango de índices especificando dónde comenzar y dónde terminar el rango.

Al especificar un rango, el valor de retorno será una nueva tupla con los elementos especificados.

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon",  
"mango")  
print(thistuple[2:5])
```

#This will return the items from position 2 to 5.

#Remember that the first item is position 0,  
#and note that the item in position 5 is NOT included

```
('cherry', 'orange', 'kiwi')
```



# Lenguaje python - Tuplas

## Rango de índices negativos

Especifique índices negativos si desea iniciar la búsqueda desde el final de la tupla:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon",  
"mango")  
print(thistuple[-4:-1])
```

#Negative indexing means starting from the end of the tuple.

#This example returns the items from index -4 (included) to index -1 (excluded)

#Remember that the last item has the index -1,

```
('orange', 'kiwi', 'melon')
```

# Lenguaje python - Tuplas

Podemos utilizar el operador `[]` debido a que las tuplas, al igual que las listas, forman parte de un tipo de objetos llamados secuencias.

Una cadena también  
es una secuencia...

```
c = "hola mundo"
c[0]    # h
c[5:]   # mundo
c[:3]   # hauo
```

# Lenguaje python - Tuplas

- Las tuplas son inmutables: Una vez creadas no se puede hacer update sobre los datos asignados inicialmente
- Tienen un tamaño fijo
- Las tuplas son más “livianas” que las listas
- Se aconsejan solo para ahorrar memoria

# Lenguaje python - Tuplas

## Cambiar valores de tupla

Una vez que se crea una tupla, no puede cambiar sus valores. Las tuplas son **inmutables**, o **inmutables** como también se las llama.

Pero hay una solución. Puede convertir la tupla en una lista, cambiar la lista y volver a convertir la lista en una tupla.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
("apple", "kiwi", "cherry")
```

# Lenguaje python - Tuplas

## Bucle a través de una tupla

Puede recorrer los elementos de la tupla utilizando un `for` bucle.

```
thistuple = ("apple", "banana", "cherry")  
for x in thistuple:  
    print(x)
```

```
apple  
banana  
cherry
```

# Lenguaje python - Tuplas

## Compruebe si el artículo existe

Para determinar si un elemento específico está presente en una tupla, use la `in` palabra clave:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
    print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

# Lenguaje python - Tuplas

## Longitud de la tupla

Para determinar cuántos elementos tiene una tupla, use el `len()` método:

```
thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))
```

```
3
```

# Lenguaje python - Tuplas

## Agregar elementos

Una vez que se crea una tupla, no puede agregarle elementos. Las tuplas son **inmutables**.

### Ejemplo

No puede agregar elementos a una tupla:

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
xList=list(thistuple)
xList.append("pera")
thistuple= tuple(xList)
print(thistuple)
```

```
('apple', 'banana', 'cherry')
('apple', 'banana', 'cherry', 'pera')
```



# Lenguaje python - Tuplas

## Crear tupla con un elemento

Para crear una tupla con un solo elemento, debe agregar una coma después del elemento; de lo contrario, Python no lo reconocerá como una tupla.

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

# Lenguaje python - Tuplas

## El constructor tuple ()

También es posible usar el constructor `tuple ()` para hacer una tupla.

Usando el método tuple () para hacer una tupla:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```

```
thistuple = tuple(("apple", "banana", "cherry"))
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

# Lenguaje python - Tuplas

## Eliminar elementos

**Nota:** no puede eliminar elementos en una tupla.

Las tuplas no se pueden **cambiar** , por lo que no puede eliminar elementos de ellas, pero puede eliminar la tupla por completo:

## Ejemplo

La `del` palabra clave puede eliminar la tupla por completo:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

# Lenguaje python - Tuplas

## Alternativa a eliminar elementos

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple)  
xList=list(thistuple)  
xList.remove("cherry")  
thistuple= tuple(xList)  
print(thistuple)
```

```
('apple', 'banana', 'cherry')  
( 'apple', 'banana')
```

# Lenguaje python - Tuplas

## Unir dos tuplas

Para unir dos o más tuplas, puede utilizar el `+` operador:

```
tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

# Lenguaje python - Tuplas

## Métodos de tupla

Python tiene dos métodos integrados que puede usar en tuplas.

Method	Description
<u>count()</u>	Returns the number of times a specified value occurs in a tuple
<u>index()</u>	Searches the tuple for a specified value and returns the position of where it was found

# Lenguaje python - Tuplas

## Ejemplo

Devuelve el número de veces que aparece el valor 5 en la tupla:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)  
x = thistuple.count(5)  
print(x)
```

2

# Lenguaje python - Tuplas

## Ejemplo

Busque la primera aparición del valor 8 y devuelva su posición:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)  
x = thistuple.index(8)  
print(x)
```

3



# Lenguaje python – set

## set

Un conjunto es una colección que **no** está **ordenada** ni **indexada**. En Python, los conjuntos se escriben con llaves.

**Nota: Los** conjuntos están desordenados, por lo que no puede estar seguro en qué orden aparecerán los elementos.

```
thisset = {"apple", "banana", "cherry"}  
print(thisset)
```

# Note: the set list is unordered, meaning: the items  
will appear in a random order.

# Refresh this page to see the change in the result.

```
{'banana', 'cherry', 'apple'}
```

# Lenguaje python - set

## Elementos de acceso

No puede acceder a los elementos de un conjunto haciendo referencia a un índice, ya que los conjuntos están desordenados y los elementos no tienen índice.

Pero puede recorrer los elementos del conjunto utilizando un for ciclo, o preguntar si un valor específico está presente en un conjunto, utilizando la palabra clave **in**.

```
thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
    print(x)
```

```
apple  
cherry  
banana
```

### Ejemplo

Compruebe si "banana" está presente en el conjunto:

```
thisset = {"apple", "banana", "cherry"}  
  
print("banana" in thisset)
```

```
True
```

# Lenguaje python - set

## Agregar elementos

Para agregar un artículo a un conjunto, use el método `add()`.

Para agregar más de un elemento a un conjunto, use el método `update()`.

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset)
```

```
{'apple', 'banana', 'orange', 'cherry'}
```

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.update(["orange", "mango", "grapes"])  
  
print(thisset)
```

```
{'apple', 'orange', 'banana', 'grapes', 'cherry', 'mango'}
```

# Lenguaje python - set

Obtenga la longitud de un conjunto

Para determinar cuántos elementos tiene un conjunto, use el método `len()`.

```
thisset = {"apple", "banana", "cherry"}  
  
print(len(thisset))  
|
```

3

# Lenguaje python - set

## Remover el artículo

Para eliminar un elemento de un conjunto, utilice método `remove()` o el método `discard()`.

```
thisset = {"apple", "banana", "cherry"}  
thisset.remove("banana")  
print(thisset)
```

```
{'cherry', 'apple'}
```

**Nota:** Si el elemento a eliminar no existe, `remove()` generará un error.

```
thisset = {"apple", "banana", "cherry"}  
thisset.discard("banana")  
print(thisset)
```

```
{'cherry', 'apple'}
```

**Nota:** Si el elemento a eliminar no existe, `discard()` **NO** generará un error.

# Lenguaje python - set

También puede utilizar el método `pop()`, para eliminar el último elemento..

Recuerde que los conjuntos están desordenados, por lo que no sabrá qué elemento se quita.

El valor de retorno del método `pop()` es el elemento eliminado.

```
thisset = {"apple", "banana", "cherry"}  
  
x = thisset.pop()  
  
print(x) #removed item  
  
print(thisset) #the set after removal
```

```
cherry  
{'apple', 'banana'}
```

# Lenguaje python - set

El método `clear()` vacía el conjunto:

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.clear()  
  
print(thisset)
```



set()

La palabra clave `del` eliminará el conjunto por completo:

```
thisset = {"apple", "banana", "cherry"}  
  
del thisset  
  
print(thisset) #this will raise an error  
because the set no longer exists
```

```
Traceback (most recent call last):  
  File "demo_set_del.py", line 5, in <module>  
    print(thisset) #this will raise an error because the set no longer exists  
NameError: name 'thisset' is not defined
```

# Lenguaje python - set

## Unir dos conjuntos

Hay varias formas de unir dos o más conjuntos en Python.

Puede usar el método `union()` que devuelve un nuevo conjunto que contiene todos los elementos de ambos conjuntos, o el método `update()` que inserta todos los elementos de un conjunto en otro:

```
set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}
```

```
set3 = set1.union(set2)  
print(set3)
```

```
set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}
```

```
set1.update(set2)  
print(set1)
```

```
{'b', 'a', 2, 3, 'c', 1}
```

```
{'c', 1, 'a', 3, 2, 'b'}
```

### **Nota:**

`union()` y `update()`  
excluirán cualquier  
artículo duplicado.



# Lenguaje python - set

## El constructor set ()

También es posible utilizar el constructor `set ()` para hacer un conjunto.

```
thisset = set(("apple", "banana", "cherry"))  
print(thisset)  
# Note: the set list is unordered, so the  
result will display the items in a random  
order.
```

```
{'cherry', 'banana', 'apple'}
```

# Lenguaje python - set

## Métodos de Set

Method	Description
<u>add()</u>	Adds an element to the set
<u>clear()</u>	Removes all the elements from the set
<u>copy()</u>	Returns a copy of the set
<u>difference()</u>	Returns a set containing the difference between two or more sets
<u>difference_update()</u>	Removes the items in this set that are also included in another, specified set
<u>discard()</u>	Remove the specified item
<u>intersection()</u>	Returns a set, that is the intersection of two other sets
<u>intersection_update()</u>	Removes the items in this set that are not present in other, specified set(s)

# Lenguaje python - set

## Métodos de Set

<code>isdisjoint()</code>	Returns whether two sets have a intersection or not
<code>issubset()</code>	Returns whether another set contains this set or not
<code>issuperset()</code>	Returns whether this set contains another set or not
<code>pop()</code>	Removes an element from the set
<code>remove()</code>	Removes the specified element
<code>symmetric_difference()</code>	Returns a set with the symmetric differences of two sets
<code>symmetric_difference_update()</code>	inserts the symmetric differences from this set and another
<code>union()</code>	Return a set containing the union of sets
<code>update()</code>	Update the set with the union of this set and others

# Lenguaje python - set

```
fruits = {"apple", "banana", "cherry"}  
  
x = fruits.copy()  
  
print(x)
```

```
{'cherry', 'banana', 'apple'}
```

```
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.difference(y)  
  
print(z)
```

```
{'cherry', 'banana'}
```

# Lenguaje python - set

```
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
x.difference_update(y)  
  
print(x)
```

```
{'cherry', 'banana'}
```

```
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.intersection(y)  
  
print(z)
```

```
{'apple'}
```

```
x = {"a", "b", "c"}  
y = {"c", "d", "e"}  
z = {"f", "g", "c"}
```

```
result = x.intersection(y, z)  
  
print(result)
```

```
{'c'}
```

# Lenguaje python - set

```
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
x.intersection_update(y)  
  
print(x)
```

```
{'apple'}
```

```
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "facebook"}  
  
z = x.isdisjoint(y)  
  
print(z)
```

```
True
```

Devuelve True si no hay elementos en el conjunto X presentes en el conjunto Y

# Lenguaje python - set

```
x = {"a", "b", "c"}  
y = {"f", "e", "d", "c", "b", "a"}  
  
z = x.issubset(y)  
  
print(z)
```

True

Devuelve True si todos los elementos establecidos X están presentes en el conjunto Y

```
x = {"f", "e", "d", "c", "b", "a"}  
y = {"a", "b", "c"}  
  
z = x.issuperset(y)  
  
print(z)
```

True

Devuelve True si todos los elementos establecidos Y están presentes en el conjunto X

# Lenguaje python - set

Devuelve un conjunto que contiene todos los elementos de ambos conjuntos, excepto los elementos que están presentes en ambos conjuntos

```
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.symmetric_difference(y)  
  
print(z)
```

```
{'cherry', 'google', 'microsoft', 'banana'}
```

Quita los elementos que están presentes en ambos conjuntos e inserta los elementos que no están presentes en ambos conjuntos

```
x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
x.symmetric_difference_update(y)  
  
print(x)
```

```
{'cherry', 'microsoft', 'banana', 'google'}
```



# Lenguaje python - diccionarios

## Diccionario

Un diccionario es una colección desordenada, modificable e indexada. En Python, los diccionarios se escriben con llaves y tienen claves y valores.

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```

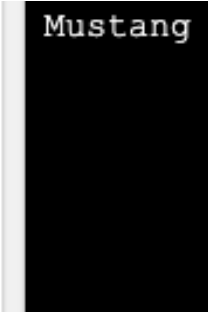
```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

# Lenguaje python - diccionarios

## Acceder a elementos

Puede acceder a los elementos de un diccionario haciendo referencia a su nombre clave, entre corchetes

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
x = thisdict["model"]  
print(x)
```

A terminal window with a black background and a light gray border. The word "Mustang" is printed in white text at the top of the window.

Mustang

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
x = thisdict.get("model")  
print(x)
```

A terminal window with a black background and a light gray border. The word "Mustang" is printed in white text at the top of the window.

Mustang

# Lenguaje python - diccionarios

## Cambiar valores

Puede cambiar el valor de un elemento específico consultando su nombre clave

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
thisdict["year"] = 2018  
  
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

# Lenguaje python - diccionarios

## Recorrer un diccionario

Puede recorrer un diccionario mediante un bucle for.

Al recorrer un diccionario, el valor de retorno son las *claves* del diccionario, pero también existen métodos para devolver los *valores* .

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict:  
    print(x)
```

```
brand  
model  
year
```

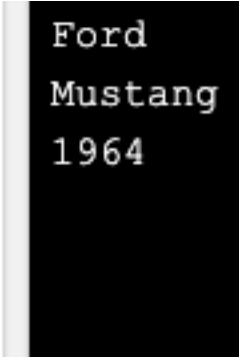
```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict:  
    print(thisdict[x])
```

```
Ford  
Mustang  
1964
```

# Lenguaje python - diccionarios

También puede usar el método `values()` para devolver valores de un diccionario

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x in thisdict.values():  
    print(x)
```



```
Ford  
Mustang  
1964
```

# Lenguaje python - diccionarios

Recorra tanto las *claves* como los *valores* , utilizando el método `items()`

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
for x, y in thisdict.items():  
    print(x, y)
```

```
brand Ford  
model Mustang  
year 1964
```

# Lenguaje python - diccionarios

Para determinar si una clave específica está presente en un diccionario, use la palabra clave **in**

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
if "model" in thisdict:  
    print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

# Lenguaje python - diccionarios

Para determinar cuántos elementos (pares clave-valor) tiene un diccionario, use la función `len()`

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
print(len(thisdict))
```

3



# Lenguaje python - diccionarios

## Agregar elementos

La adición de un elemento al diccionario se realiza utilizando una nueva clave de índice y asignándole un valor

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

# Lenguaje python - diccionarios

## Eliminar elementos

Existen varios métodos para eliminar elementos de un diccionario

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

# Lenguaje python - diccionarios

El método `popitem()` elimina el último elemento insertado (en las versiones anteriores a la 3.7, se elimina un elemento aleatorio en su lugar)

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.popitem()  
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

# Lenguaje python - diccionarios

La palabra clave **del** elimina el elemento con el nombre de clave especificado

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
del thisdict["model"]  
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

La palabra clave **del** también puede eliminar el diccionario por completo

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error  
because "thisdict" no longer exists.
```

```
Traceback (most recent call last):  
  File "demo_dictionary_del3.py", line 7, in <module>  
    print(thisdict) #this will cause an error because "thisdict" no longer exists  
NameError: name 'thisdict' is not defined
```

# Lenguaje python - diccionarios

## Copiar un diccionario

No puede copiar un diccionario simplemente escribiendo `dict2 = dict1`, porque: `dict2` solo será una *referencia* a `dict1`, y los cambios realizados en `dict1` automáticamente también se realizarán en `dict2`.

Hay formas de hacer una copia, una forma es utilizar el método de diccionario integrado `copy()`

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

# Lenguaje python - diccionarios

Otra forma de hacer una copia es utilizar la función incorporada dict()

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

# Lenguaje python - diccionarios

## Diccionarios anidados

Un diccionario también puede contener muchos diccionarios, esto se llama diccionarios anidados

```
myfamily = {  
    "child1" : {  
        "name" : "Emil",  
        "year" : 2004  
    },  
    "child2" : {  
        "name" : "Tobias",  
        "year" : 2007  
    },  
    "child3" : {  
        "name" : "Linus",  
        "year" : 2011  
    }  
}  
  
print(myfamily)
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

# Lenguaje python - diccionarios

fromkey() crea un diccionario con 3 claves, todas con el valor 0

```
x = ('key1', 'key2', 'key3')
y = 0

thisdict = dict.fromkeys(x, y)

print(thisdict)
```

```
['key1': 0, 'key2': 0, 'key3': 0]
```



# Lenguaje python - diccionarios

Method	Description
<u>clear()</u>	Removes all the elements from the dictionary
<u>copy()</u>	Returns a copy of the dictionary
<u>fromkeys()</u>	Returns a dictionary with the specified keys and value
<u>get()</u>	Returns the value of the specified key
<u>items()</u>	Returns a list containing a tuple for each key value pair
<u>keys()</u>	Returns a list containing the dictionary's keys
<u>pop()</u>	Removes the element with the specified key
<u>popitem()</u>	Removes the last inserted key-value pair
<u>setdefault()</u>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<u>update()</u>	Updates the dictionary with the specified key-value pairs
<u>values()</u>	Returns a list of all the values in the dictionary

# Lenguaje python – Estructuras de decisión

## Condiciones de Python y declaraciones If

Python admite las condiciones lógicas habituales de las matemáticas:

- Es igual a: `a == b`
- No es igual a: `a != B`
- Menor que: `a < b`
- Menor o igual a: `a <= b`
- Mayor que: `a > b`
- Mayor o igual a: `a >= b`

Estas condiciones se pueden utilizar de varias formas, más comúnmente en "sentencias if" y bucles.

Una "instrucción if" se escribe utilizando la palabra clave `if` .

# Lenguaje python - Estructuras de decisión

```
a = 33
b = 200

if b > a:
    print("b is greater than a")
```

```
b is greater than a
```

## Elif

La palabra clave **elif** es la forma de pitón de decir "si las condiciones anteriores no eran verdaderas, entonces pruebe esta condición"

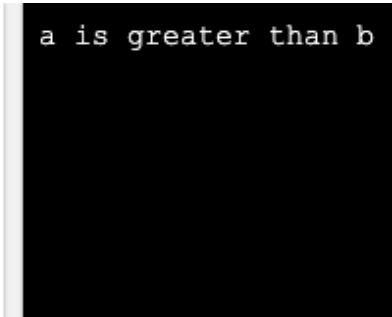
```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

```
a and b are equal
```

# Lenguaje python - Estructuras de decisión

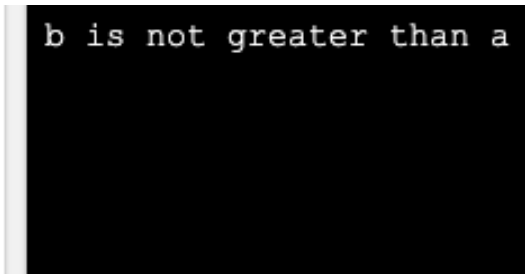
La palabra clave `else` captura cualquier cosa que no sea detectada por las condiciones anteriores

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

A terminal window with a black background and white text. The text "a is greater than b" is displayed on the first line.

a is greater than b

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

A terminal window with a black background and white text. The text "b is not greater than a" is displayed on the first line.

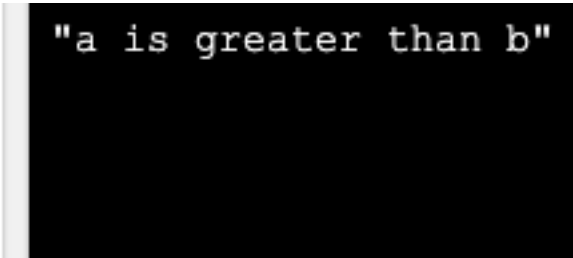
b is not greater than a

# Lenguaje python - Estructuras de decisión

Si solo tiene una instrucción para ejecutar, puede ponerla en la misma línea que la instrucción **if**

```
a = 200
b = 33

if a > b: print("a is greater than b")
```

A terminal window with a black background and white text displaying the output of the Python code: "a is greater than b".

"a is greater than b"

Si solo tiene una declaración para ejecutar, una para if y otra para otra, puede ponerlas todas en la misma línea

```
a = 2
b = 330

print("A") if a > b else print("B")
```

A terminal window with a black background and white text displaying the output of the Python code: B.

B

# Lenguaje python - Estructuras de decisión

Esta técnica se conoce como **Operadores ternarios** o **Expresiones condicionales** .

También puede tener varias declaraciones else en la misma línea:

```
a = 330  
b = 330
```

```
print("A") if a > b else print("=") if a == b else print("B")
```

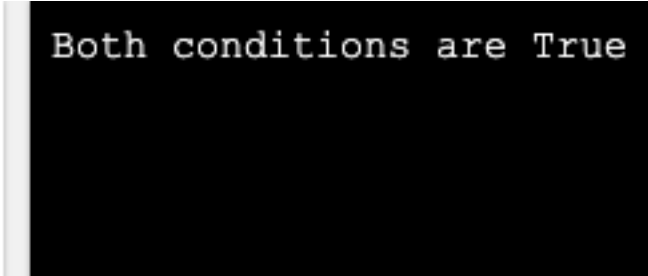


# Lenguaje python - Estructuras de decisión

## Y - and

La palabra clave `y` es un operador lógico y se usa para combinar declaraciones condicionales

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

A screenshot of a terminal window with a black background and white text. The text displayed is "Both conditions are True".

Both conditions are True

# Lenguaje python - Estructuras de decisión

## O - or

La palabra clave **or** es un operador lógico y se usa para combinar declaraciones condicionales

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```



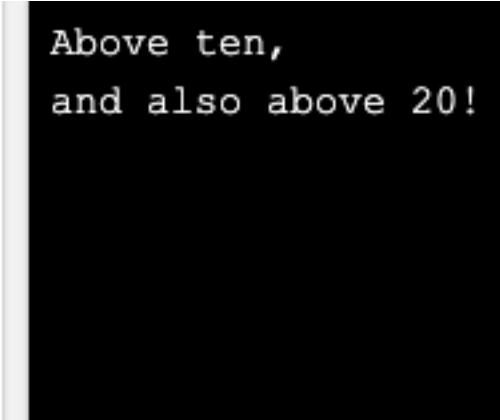
# Lenguaje python - Estructuras de decisión

## if anidado

Puede tener declaraciones `if` dentro de declaraciones `if`, esto se llama declaraciones *anidadas* `if`

```
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```



```
Above ten,
and also above 20!
```

# Lenguaje python - Estructuras de decisión

## La declaración `pass`

Las declaraciones `if` no pueden estar vacías, pero si por alguna razón tiene una declaración `if` sin contenido, introdúzcala `pass` para evitar errores

```
a = 33  
b = 200
```

```
if b > a:  
    pass
```

```
# having an empty if statement  
like this, would raise an error  
without the pass statement
```

# Lenguaje python - Estructuras de repetición

## Bucles de Python

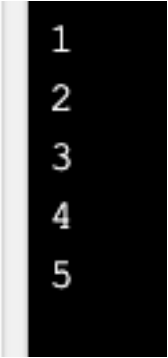
Python tiene dos comandos de bucle primitivos:

- `while` bucles
- `for` bucles

### El bucle `while`

Con el bucle `while` podemos ejecutar un conjunto de instrucciones, siempre y cuando se cumpla una condición

```
i = 1
while i < 6:
    print(i)
    i += 1
```



```
1
2
3
4
5
```

# Lenguaje python - Estructuras de repetición

Con la sentencia `break` podemos detener el ciclo incluso si la condición while es verdadera

```
i = 1
while i < 6:
    print(i)
    if (i == 3):
        break
    i += 1
```



```
1
2
3
```

# Lenguaje python - Estructuras de repetición

Con la declaración `continue` podemos detener la iteración actual y continuar con la siguiente

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

# Note that number 3 is missing in the result



1  
2  
4  
5  
6

# Lenguaje python - Estructuras de repetición

Con la instrucción `else` podemos ejecutar un bloque de código una vez que la condición ya no es verdadera

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
```

# Lenguaje python - Estructuras de repetición

## Bucles **for**

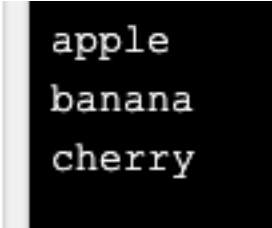
Un bucle **for** se utiliza para iterar sobre una secuencia (que es una lista, una tupla, un diccionario, un conjunto o una cadena).

Esto es menos parecido a la palabra clave **for** en otros lenguajes de programación y funciona más como un método iterador como se encuentra en otros lenguajes de programación orientados a objetos.

Con el bucle **for** podemos ejecutar un conjunto de sentencias, una vez para cada elemento de una lista, tupla, conjunto, etc.

# Lenguaje python - Estructuras de repetición

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```



```
apple  
banana  
cherry
```

Incluso las cadenas son objetos iterables, contienen una secuencia de caracteres

```
for x in "banana":  
    print(x)
```



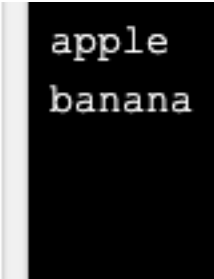
```
b  
a  
n  
a  
n  
a
```



# Lenguaje python - Estructuras de repetición

Con la instrucción `break` podemos detener el ciclo antes de que haya pasado por todos los elementos

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)  
    if x == "banana":  
        break
```



```
apple  
banana
```

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    if x == "banana":  
        break  
    print(x)
```

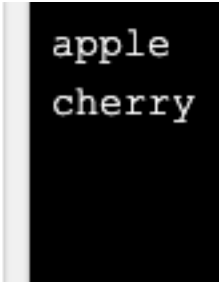


```
apple
```

# Lenguaje python - Estructuras de repetición

Con la declaración `continue` podemos detener la iteración actual del ciclo y continuar con la siguiente

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    if x == "banana":  
        continue  
    print(x)
```



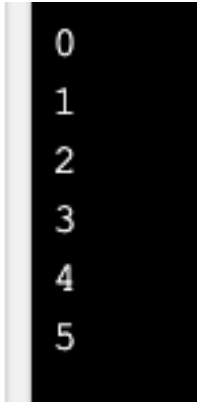
```
apple  
cherry
```

# Lenguaje python - Estructuras de repetición

## La función `range()`

La función `range()` devuelve una secuencia de números, comenzando desde 0 de forma predeterminada, se incrementa en 1 (de forma predeterminada) y termina en un número especificado

```
for x in range(6):  
    print(x)
```



```
0  
1  
2  
3  
4  
5
```

# Lenguaje python - Estructuras de repetición

La función `range()` tiene por defecto 0 como valor inicial, sin embargo, es posible especificar el valor inicial agregando un parámetro: `range(2, 6)`, que significa valores de 2 a 6 (pero sin incluir 6)

```
for x in range(2, 6):  
    print(x)
```



```
2  
3  
4  
5
```

# Lenguaje python - Estructuras de repetición

La función `range()` tiene como valor predeterminado para incrementar la secuencia en 1, sin embargo, es posible especificar el valor de incremento agregando un tercer parámetro: `range(2, 30, 3)`

```
for x in range(2, 30, 3):  
    print(x)
```

```
2  
5  
8  
11  
14  
17  
20  
23  
26  
29
```

# Lenguaje python - Estructuras de repetición

La palabra clave **else** en un bucle **for** especifica un bloque de código que se ejecutará cuando finalice el bucle

```
for x in range(6):  
    print(x)  
else:  
    print("Finally finished!")
```

```
0  
1  
2  
3  
4  
5  
Finally finished!
```

# Lenguaje python - Estructuras de repetición

Un bucle anidado es un bucle dentro de un bucle.

El "bucle interno" se ejecutará una vez por cada iteración del "bucle externo"

```
adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]  
  
for x in adj:  
    for y in fruits:  
        print(x, y)
```

```
red apple  
red banana  
red cherry  
big apple  
big banana  
big cherry  
tasty apple  
tasty banana  
tasty cherry
```

# Lenguaje python - Estructuras de repetición

Los bucles **for** no pueden estar vacíos, pero si por alguna razón tiene uno sin contenido, ingrese `pass` en la declaración para evitar errores

```
for x in [0, 1, 2]:  
    pass
```

```
# having an empty for loop like this,  
would raise an error without the pass  
statement
```





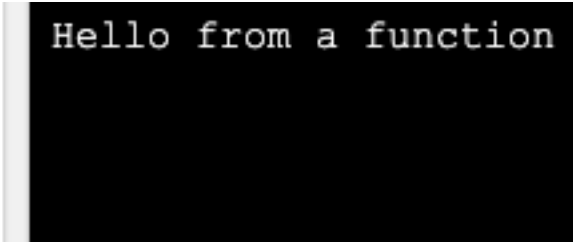
# Lenguaje python - Funciones

Una función es un bloque de código que solo se ejecuta cuando se llama.

Puede pasar datos, conocidos como parámetros, a una función.

Una función puede devolver datos como resultado.

```
def my_function():  
    print("Hello from a function")  
  
my_function()
```

A screenshot of a terminal window with a black background. The text "Hello from a function" is displayed in a light gray, monospaced font.

```
Hello from a function
```

# Lenguaje python - Funciones

## Argumentos

La información se puede pasar a funciones como argumentos.

Los argumentos se especifican después del nombre de la función, entre paréntesis.

Puede agregar tantos argumentos como desee, solo sepárelos con una coma.

El siguiente ejemplo tiene una función con un argumento (fname). Cuando se llama a la función, pasamos un nombre, que se usa dentro de la función para imprimir el nombre completo

```
def my_function(fname):  
    print(fname + " Refsnes")  
  
my_function("Emil")  
my_function("Tobias")  
my_function("Linus")
```

```
Emil Refsnes  
Tobias Refsnes  
Linus Refsnes
```

# Lenguaje python - Funciones

## ¿Parámetros o argumentos?

Los términos *parámetro* y *argumento* se pueden usar para lo mismo: información que se pasa a una función.

Desde la perspectiva de una función:

Un parámetro es la variable que aparece entre paréntesis en la definición de la función.

Un argumento es el valor que se envía a la función cuando se llama.

# Lenguaje python - Funciones

## Número de argumentos

Por defecto, se debe llamar a una función con el número correcto de argumentos. Lo que significa que si su función espera 2 argumentos, debe llamar a la función con 2 argumentos, ni más ni menos

```
def my_function(fname, lname):  
    print(fname + " " + lname)  
  
my_function("Emil", "Refsnes")
```

A screenshot of a terminal window with a black background. The text "Emil Refsnes" is displayed in a light blue or cyan monospaced font, representing the output of the print statement in the code block to the left.

Emil Refsnes

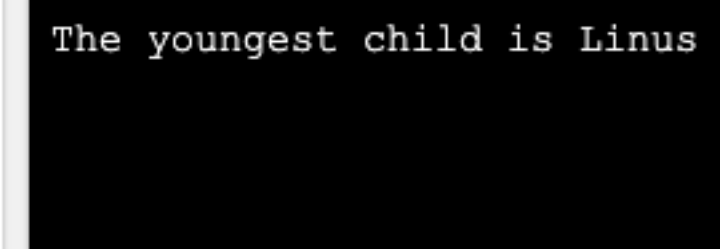
# Lenguaje python - Funciones

## Argumentos arbitrarios, \* argumentos

Si no sabe cuántos argumentos se pasarán a su función, agregue un `*` antes del nombre del parámetro en la definición de la función.

De esta forma, la función recibirá una *tupla* de argumentos y podrá acceder a los elementos en consecuencia

```
def my_function(*kids):  
    print("The youngest child is " + kids[2])  
  
my_function("Emil", "Tobias", "Linus")
```

A screenshot of a terminal window with a black background and white text. The text displayed is "The youngest child is Linus", which is the output of the Python function defined in the previous block.

```
The youngest child is Linus
```

# Lenguaje python - Funciones

## Argumentos usando palabras clave

También puede enviar argumentos con la sintaxis *clave = valor* .

De esta manera, el orden de los argumentos no importa

```
def my_function(child3, child2, child1):  
    print("The youngest child is " + child3)  
  
my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

```
The youngest child is Linus
```

# Lenguaje python - Funciones

Si no sabe cuántos argumentos de palabras clave se pasarán a su función, agregue dos asteriscos `**` antes del nombre del parámetro en la definición de la función.

De esta forma, la función recibirá un *diccionario* de argumentos y podrá acceder a los elementos en consecuencia

```
def my_function(**kid):  
    print("His last name is " + kid["lname"])  
  
my_function(fname = "Tobias", lname = "Refsnes")
```

```
His last name is Refsnes
```

# Lenguaje python - Funciones

## Valor de parámetro predeterminado

El siguiente ejemplo muestra cómo utilizar un valor de parámetro predeterminado

```
def my_function(country = "Norway"):  
    print("I am from " + country)  
  
my_function("Sweden")  
my_function("India")  
my_function()  
my_function("Brazil")
```

```
I am from Sweden  
I am from India  
I am from Norway  
I am from Brazil
```

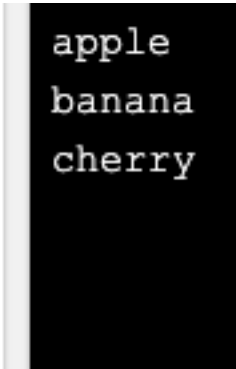


# Lenguaje python - Funciones

Puede enviar cualquier tipo de datos de argumento a una función (cadena, número, lista, diccionario, etc.), y se tratará como el mismo tipo de datos dentro de la función.

Por ejemplo, si envía una Lista como argumento, seguirá siendo una Lista cuando llegue a la función:

```
def my_function(food):  
    for x in food:  
        print(x)  
  
fruits = ["apple", "banana", "cherry"]  
  
my_function(fruits)
```



apple  
banana  
cherry

# Lenguaje python - Funciones

## Valores devueltos

Para permitir que una función devuelva un valor, use la return declaración

```
def my_function(x):  
    return 5 * x  
  
print(my_function(3))  
print(my_function(5))  
print(my_function(9))
```

```
15  
25  
45
```

# Lenguaje python - Funciones



# Lenguaje python - Funciones

